**CSCE606 - Team Anonymous**

**Biology Learning Games: StepStone Research**

**2019/12/10**

**Team Members**

J Rivera

Aleksandr Earnest

Mahin Ramezani

Cesar Garcia

Chandan Acharya

Neeraj Aduri

Iteration 1:

1. The overall flow of the StepStone framework is constructed using JSON files that store information about the placement of images, documents, A/V elements, text and most importantly, the associated mini-app for the step.
2. Each step can only store one concurrent mini-app, which links to an app in the apps folder by the name. The application must have an index HTML file in the directory.
3. The deployment server did not have NPM installed, thus when the application was attempted to be accessed, it would provide a blank screen. The fix to this, however, is to modify the paths the application uses to find the minified CSS, JavaScript, and HTML files. Perhaps even modify the application to be a Single Page Application.
4. StepStone paths, the overall flow of an application, is broken down into steps. Each step is defined by a JSON file, which shows the path from the current step to other steps, and what components are seen on screen. The names within the directory are “draggybox[number of cell].json”. The first and last steps are always draggybox1 and draggybox2 respectively.
5. The overall directory for a StepStone application is broken down by three folders, “apps”, “json” and “media”. The apps folder contains all references to mini-apps the path could use. The json folder contains the contents to the flow of the path and what data is supposed to appear in the steps. Finally, the media folder actually stores all media used by the steps and can be referenced directly from there like if it is in the same folder as the JSONs.
6. There are two different modes to paths within StepStone, that being case study and procedural. The case study model allows for returning to previous steps, whereas the procedural form does not.
7. Deployment of applications in the interface is the same as deployment on any other server but requires the root to be in the apps folder otherwise, the StepStone path cannot see it.
8. There can be as many applications in the apps folder, but only one can be linked to a StepStone step at a time.

Iteration 2:

Through more research into StepStone and how it interacts with “Mini-Apps”, the following information was observed:

1. The structure json provides the flow of the module listed in the path, such as the clickable steps that can be jumped to from the main menu for the module. In this file, to show these steps, you must define the x and y positions to where they are displayed, the node’s ID, completion status, node label and the output pathing along with the ideal path for the steps.

2. Extra data can be loaded into the path such as the metadata of the version type, path type, title and path time stamps, along with media files, author listings and attributions. This is done through the pathParams json.

3. Media files can be preloaded as references and linked to certain steps and even the title sequence by modifying the mediaList json. These are broken down into three types, documents, A/V and Images, and each require the file name, type, variant and its usage within the path.

4. Similar to the media list, the clipBoardList json can be loaded with files for the clipboard for each of the steps.

5. Each of the steps that define the body of the module is defined as a “draggybox.” In each box, the content of step is defined as individual values, which are loaded and tacked on in order of where they are assigned (for instance, if a textbox is defined before an image, the image would be lower in the step than the textbox). These values include the title, text, media and the “Mini-App”. There are also sections for how the routing from the step is done, as the type of output it is requiring, the behavior in the way of redirect or step forward, the target node and any text message that should appear before it is routed.

6. The module is contained in the activityLib folder under the name its given, and is broken up into 3 main sections, namely the apps, json and media folders. In the apps folder, all mini applications used by the module is contained within the main index html visible in the first folder. The json folder contains all of the notable jsons mentioned above. Finally, all of the media referenced by the module through the jsons are stored in the media folder.

7. The actual path is defined in a singular json file nested in the courseLib folder under the name of the path you created in the json folder. Inside this file, the version, target language (denoted with a 2-letter abbreviation), the title, time stamp, overview and activities (modules) are listed. The modules require the type of module it is (most used is Course Module Group), the variant (usually Basic) and the module itself. This module can be set with multiple values, but mainly requires the ID, which is the title of the folder it links to in the activityLib folder, the type of module it is, variant and any prerequisite criteria (i.e. if another module is required to be completed).

8. The json encoding is the standard encoding type, and thus can be used to encode special characters, boldness, fonts and other text parameters into the messages.

9. For a path to exist in the architecture, the courseLib folder for the path must exist, otherwise it will lead the StepStone architecture to remain blank.

10. The url of the StepStone path is the url of any other path, but the “resources” flag is set to the name of the path verbatim.

11. For StepStone to recognize the names of folders, it is advisable to not use spaces, but to use camel case or underscores, as it treats every word as a separate parameter.

12. Each mini application used must be isolated to one url page, as StepStone does not perform rerouting, thus leaves certain applications blank after a reroute is performed. As such, using the cache or session tokens are advisable to keep progress, and modify the page with what elements are visible. This particular fact is the reason why previous attempts using React failed, as the application required rerouting when an action was taken, which caused the page to white itself out (recommended to use JQuery for this reason).

Iteration 3:

Now that we have the ability to deploy static web applications onto StepStone, we have observed the following:

1. Caching Issues: While deploying updated version of the games, some team members noticed that they could still see the old versions of the application running on their web browser. This led us to believe that the browser is caching assets (i.e., HTML, CSS, JavaScript files) and does not re-fetch files, since it does not know if new versions are available. We will have to add some cache-busting strategies in order for StepStone to always fetch updated files from the FTP server.
2. HTML Issue Handling: During the process of integrating the StepStone mini-app step, it has been determined that any error or inaccessible files to the StepStone interface when generating the HTML will mirror the error in the developer mode of any browser, which can leak some information about the internal file system of the deployment server. It is advised to run a test before mass deployment to catch any of these issues, as this could be a possible security leak.
3. Media files are not affected by Media List JSON: The expected functionality of the Media List JSON is to validate which files can be used in the StepStone environment, however, files outside of the ones listed in the Media List can be used without repercussions. This is unknown if this is a bug or an intended feature, but it is worth noting.

Iteration 4:

1. Upon integrating into step stone we were faced with an unforeseen issue. Step Stone is requiring us to use an iframe resizer class. This proved to be an issue with the hard margins we had used to develop the games locally. We found out that we had to use relative locations.  The location of the resizer for future use is: <https://github.com/davidjbradshaw/iframe-resizer>